



WxCam 1.3

A Simple Java-Based Application for Gathering and Displaying Loops of Webcam Imagery to Help National Weather Service Forecasters Maintain a High Level of Situational Awareness

NATIONAL WEATHER SERVICE FORECAST OFFICE, SEATTLE WA

May 2, 2013

Jay A. Albrecht

Lead Forecaster

WxCam 1.3

A Simple Java-Based Application for Gathering and Displaying Loops of Webcam Imagery to Help National Weather Service Forecasters Maintain a High Level of Situational Awareness

Overview:

Weather enthusiasts, the tourism industry, transportation departments, air quality agencies, and other groups have made a large number of images from web cameras available to the public over the internet. Most of the imagery is updated every few minutes and is available to the general public in near real time. Since many of the pictures show sky, weather, sea, airport, or roadway conditions, the imagery is a valuable resource to the National Weather Service (NWS). Most NWS forecasters already use their favorite web browser to look at live web camera imagery in order to view current weather and road conditions.

Many NWS forecast offices have mounted large-screen television displays in their workspaces. These displays are used to monitor local news and weather reports during high impact events. WxCam 1.3 is a simple Java-based computer tool that can be used to gather and display loops of user-specified near-real-time web camera images. The program's display is optimized for display on a 1080p high definition television set (Figure 1), but it can be resized for use on a computer or television screen of just about any size. Imagery is fetched using a background process in order to eliminate interference with the continuously looping and updating display. The program is capable of holding 9 loops – each which can contain 25 images. Users can easily switch loops, increase or decrease loop speed, forward or backward single step through loops, pause loops, and resume looping. Loops are easy to build and maintain.



FIGURE 1. DR. DAVID TITLEY, FORMER NOAA DEPUTY DIRECTOR FOR OPERATIONS, DISCUSSES THE IMPORTANCE OF MAINTAINING SITUATIONAL AWARENESS (SA) WITH THE STAFF WHILE STANDING NEXT TO NWS SEATTLE'S SA DISPLAY UNIT (JULY, 2012).

Installing WxCam:

WxCam is distributed as a Windows Setup File (.msi). Contact your regional DSS focal point or the author (jay.albrecht@noaa.gov) to get your copy. This program has been reviewed by SFD, and the Western Region System Owner has approved its use. NWS offices shall have a local administrator install the software, and the program shall be added to the local office software inventory list.

Before installing WxCam, make sure that you have version 1.7 of the Java Run Time Environment (jre-1.7) or higher installed on your system. To check, open the Windows Control Panel, click the Java icon, then click the Java tab on the Java Control Panel. In the Java Runtime Environment Settings panel, click the View button. The number under the platform section needs to be 1.7 or higher; if it isn't, download and install the latest version from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. You will need administrator rights to install the Java Run-Time Environment.

You are now ready to install WxCam. Run the setup file and follow the instructions. By default, WxCam is installed to C:\ProgramFiles (x86)\NOAA National Weather Service\WxCam. You can change the target directory or drive as appropriate for your office's needs.

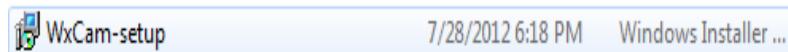


FIGURE 2. WEATHERCAM SETUP FILE.



FIGURE 3. SETUP APPLICATION WELCOME SCREEN

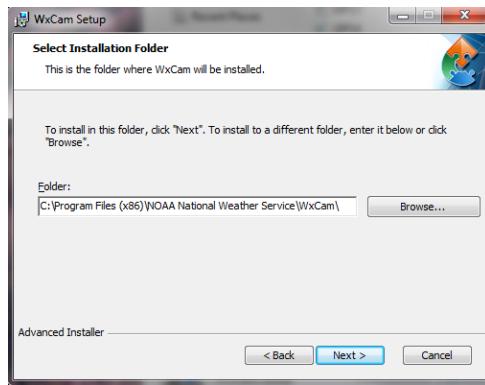


FIGURE 4. INSTALLATION FOLDER SELECTION.

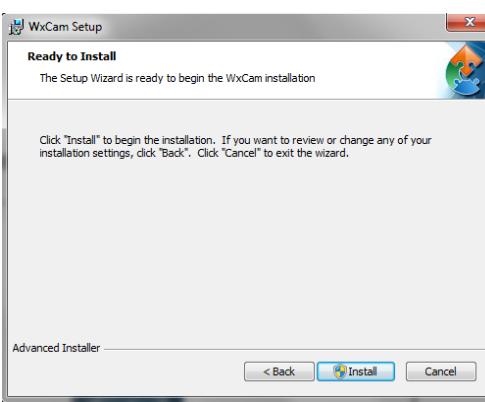


FIGURE 5. CLICK INSTALL TO PROCEED.

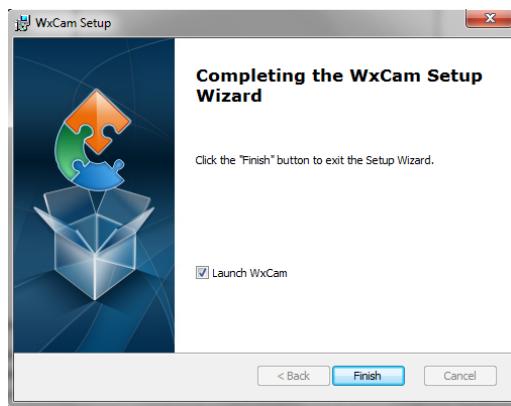


FIGURE 6. CLICK FINISH TO COMPLETE INSTALLATION.

The installation wizard will automatically add a desktop icon and a start menu program icon for you.

Configuring WxCam:

The program's configuration file, named "WxCamConfig.MCF" is placed into the directory in which you install WxCam 1.3. This configuration file is a simple custom-made database that stores your loop names, URLs of interest, picture names, and loop properties.

The program comes configured for the NWS Seattle Forecast Office. Before making changes to the configuration file, it may be useful to take the program for a quick spin. Open the program by double-clicking the WxCam icon. When the program opens, a simple black-blank screen is displayed with nothing more than a program title bar and a menu bar on the top and a status bar on the bottom. Make sure that the "numlock" key on the NUMPAD is in the on position, then click CTRL+D to start the display loop; updated imagery will begin filling the loops in the background as the loop displays. Let the program run for a few minutes, then, press the numbers (1-9) to change loops. Press the + key on the keyboard's NUMPAD to increase loop speed and the – key on the NUMPAD to decrease loop speed. Press "." to pause the loop and the NUMPAD "ENTER" key to resume the loop. If you press the / key on the NUMPAD, you can single step backward, while if you press the * key on the NUMPAD, you can single step forward. Press the NUMPAD "ENTER" key again to resume looping. You'll see live camera loops for the beautiful Seattle CWA. As images are displayed, notice the name of the image displayed on the status bar at the bottom of the screen (see Figure 7).

You can make the window larger or smaller by dragging the application's frame border; images will be resized while maintaining aspect ratio and resolution as the window is enlarged or compressed.

If imagery is not updating, press CTRL+R to stop and restart the background process that fetches imagery. It can stall at times (resulting in imagery on the screen not updating). This problem is quite infrequent (once a week, or less, on average here in Seattle).

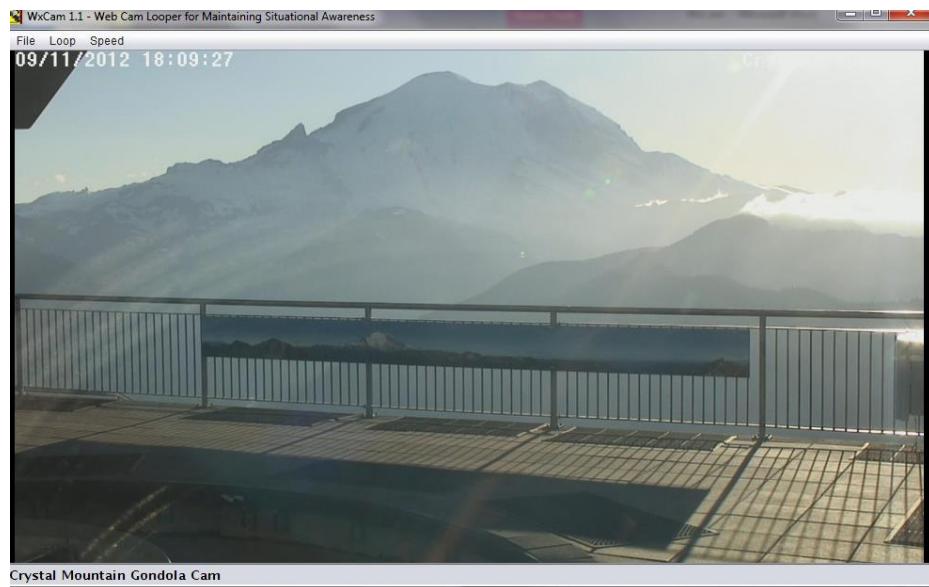


FIGURE 7. SAMPLE WEB CAM IMAGE DISPLAY.

NWS Seattle uses all 9 loops, and most loops use all of their allotment of 25 images each. Loop 1 has an assortment of images from around the Seattle NWS CWA, Loop 2 is used for aviation-related images, and Loop 3 is used to display marine-related images. Loops 4 through 9 contain images for various geographic sectors (4 is used for the north interior, 5 is for the Central Puget Sound Convergence Zone area, 6 is used for the Seattle-Bellevue metro area, 7 is used for the southwest interior region, 8 is used for the Olympic Peninsula, and 9 is used for the Cascade Mountains). Of course, your office's setup will vary depending on the number of cameras available in your area and your needs.

Images that are downloaded from the Internet are stored on your local disk. In order for the program to determine whether or not a new image needs to be downloaded for a URL of interest, the age of a previously downloaded image for a given loop and frame is checked. A new image will only be downloaded if one for that loop number and frame number has not been downloaded in 5 minutes or more. This is done to reduce internet traffic use by this program and to reduce the impact on the server providing the image. Images are also stored on disk to make the program more robust. While the images may actually be jpg-formatted images, they are stored with a CMR file extension and with the naming convention "L#F&&.CMR", where # is the loop number (0-9) and && is the frame number (00-24). If you are using all possible loops and frames, it is possible that 10-30 MB of imagery will be stored in your installation directory. The amount of storage may even be higher if all of the downloaded images are very high resolution. While there is no need to do so, you can delete any of the .CMR files from your installation directory when the program is not in use. The files will be downloaded when the program starts. The amount of data downloaded will depend on your configuration.

Now that you have an idea of how the application runs, you are ready to configure the loops for your forecast area. Click CTRL+C to display the configuration form (see Figure 8).

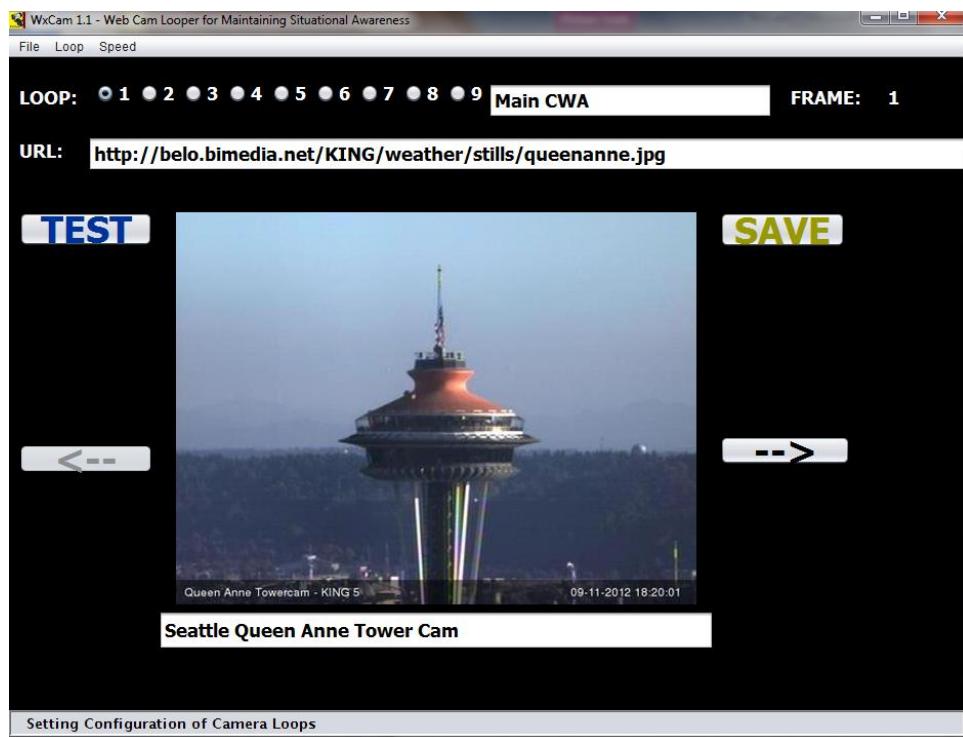


FIGURE 8. WxCAM CONFIGURATION FORM. NOTE THAT THE PICTURE WILL BE BLANK UNTIL THE "TEST" KEY IS PRESSED AND THE IMAGE IS SUCCESSFULLY DOWNLOADED. IF THERE IS A PROBLEM DOWNLOADING THE IMAGE, AN ERROR MESSAGE WILL BE DISPLAYED ON THE STATUS BAR AT THE BOTTOM OF THE PROGRAM.

The WxCam configuration process is straight forward and intuitive.

Use the “Loop” radio buttons to select a loop to edit. The name of the loop is displayed to the right of the Loop radio buttons. A loop must have a name in the Loop name edit box; if one is not there, the loop is considered empty and will not display.

Each frame has a URL and a name. The URL edit box (shown above the TEST button) holds the URL of the image to be displayed. The URL edit box must contain the entire URL name (including the http://). The name you give to the frame is displayed in the edit box below the image pane. A display frame can be any web based jpg image (some other types may work as well). You can include web based satellite images, static radar images, gridded forecast displays and other files if they are in jpg format in addition to web camera images. The easiest way to create a frame is to find the web site of an image you would like to display using your favorite browser; right click on the image and select “Copy Image Location”. Then click the URL edit box and type CTRL V to paste the URL of the image into it. (See Figure 9 and Figure 10).

Click the TEST Button (Figure 11) to ensure that the image can be displayed by this program (some can’t). If you are satisfied with your results, type a new name for the image in the camera name edit box below the image (Figure 12). Finally, click SAVE to save the entire configuration file to disk.

When you are done with this first frame, you can go to the next frame by clicking the right arrow button (Figure 13) – then repeat the process. If you click a different LOOP radio button, it will transport your form to Frame 1 of that loop, and the left arrow button will be disabled (Figure 14).

It is highly recommended that you plan out your frame and loop placement before going through the configuration process. After using the application for a while, you’ll notice that some images may not update or will become dead links. When this occurs, you can replace the image with another by using the Configuration form.

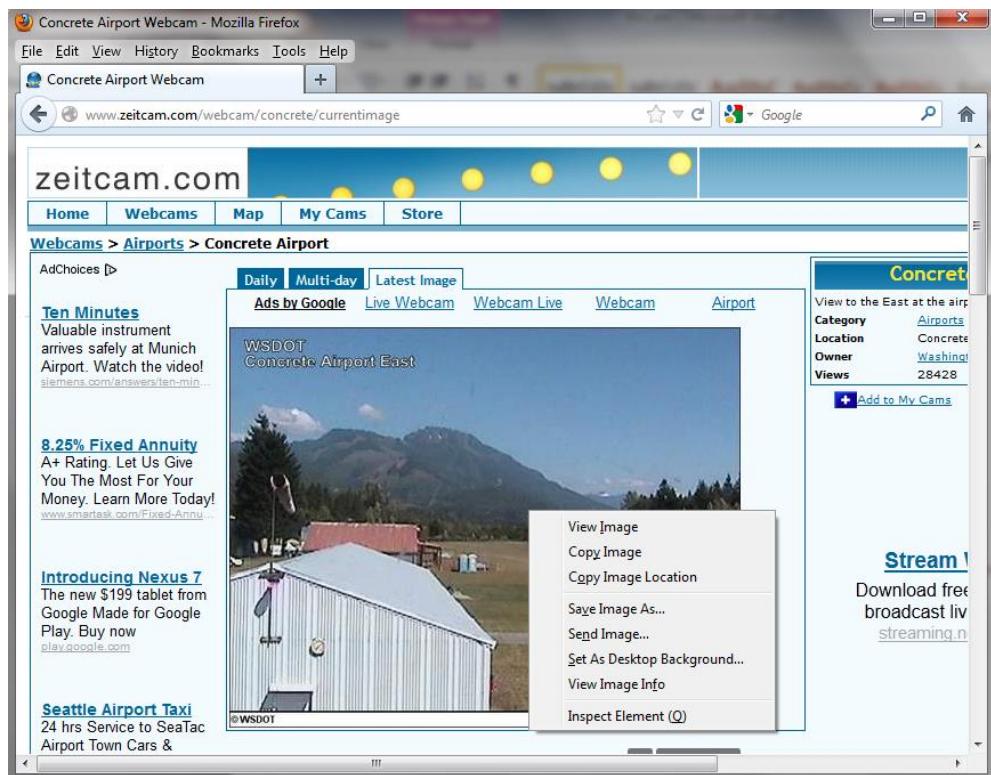


FIGURE 9. RIGHT-CLICK ON A LIVE WEBCAM IMAGE TO COPY IMAGE LOCATION. THIS WILL COPY A URL TO THE IMAGE INTO THE WINDOWS CLIPBOARD.

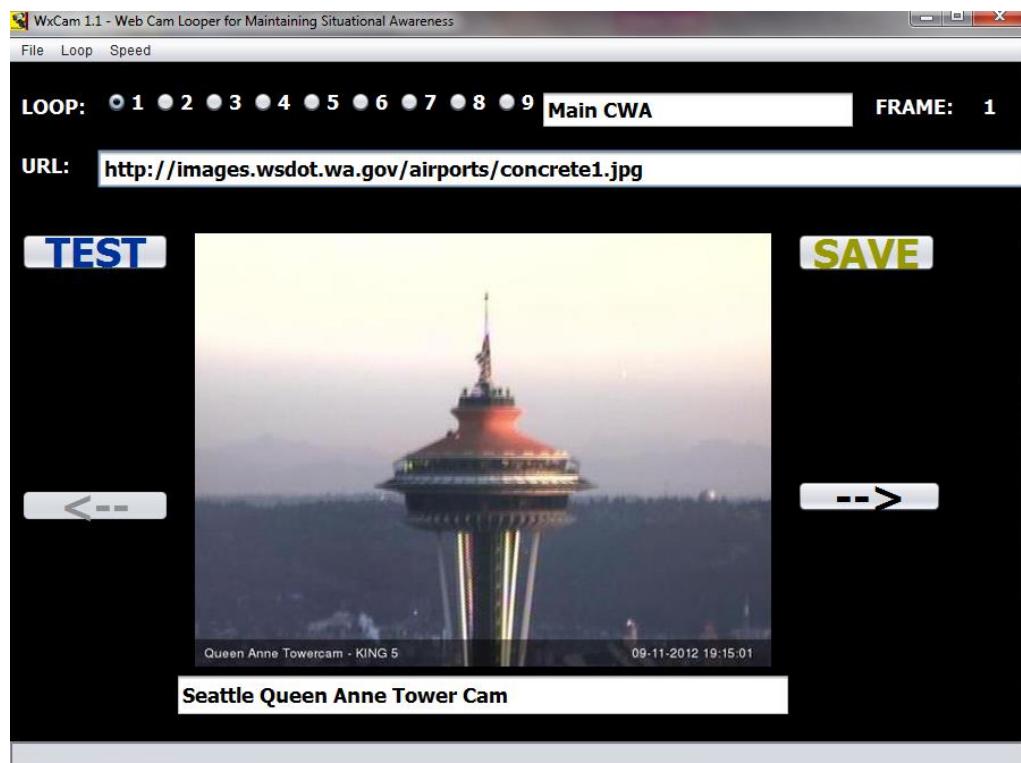


FIGURE 10. CLICK ON THE URL TEXT BOX THEN PRESS CTRL V TO PASTE IN THE URL OF THE IMAGE YOU WOULD LIKE TO USE.

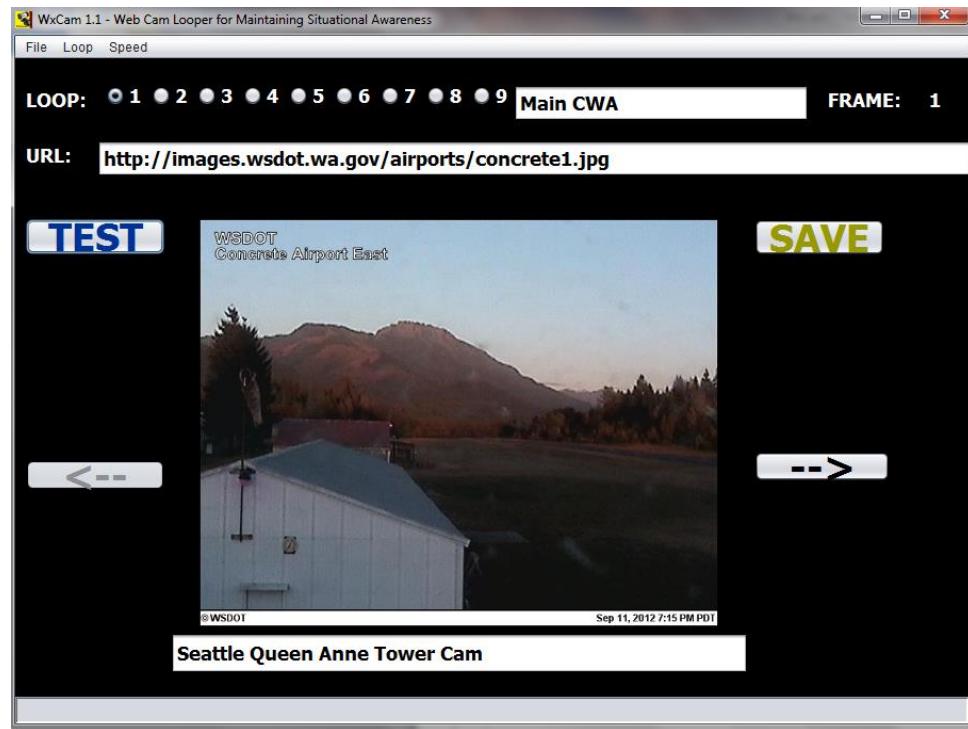


FIGURE 11. RESULTS AFTER PASTING LINK AND CLICKING TEST

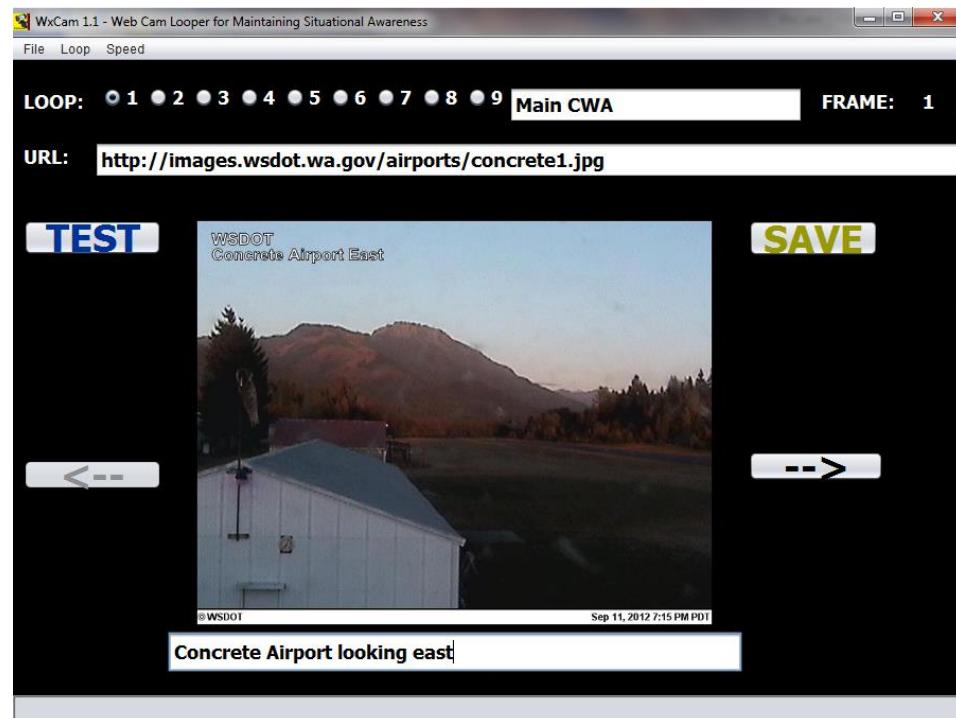


FIGURE 12. TYPE A NEW NAME FOR THE CAMERA BELOW THE IMAGE. CLICK SAVE WHEN SATISFIED.

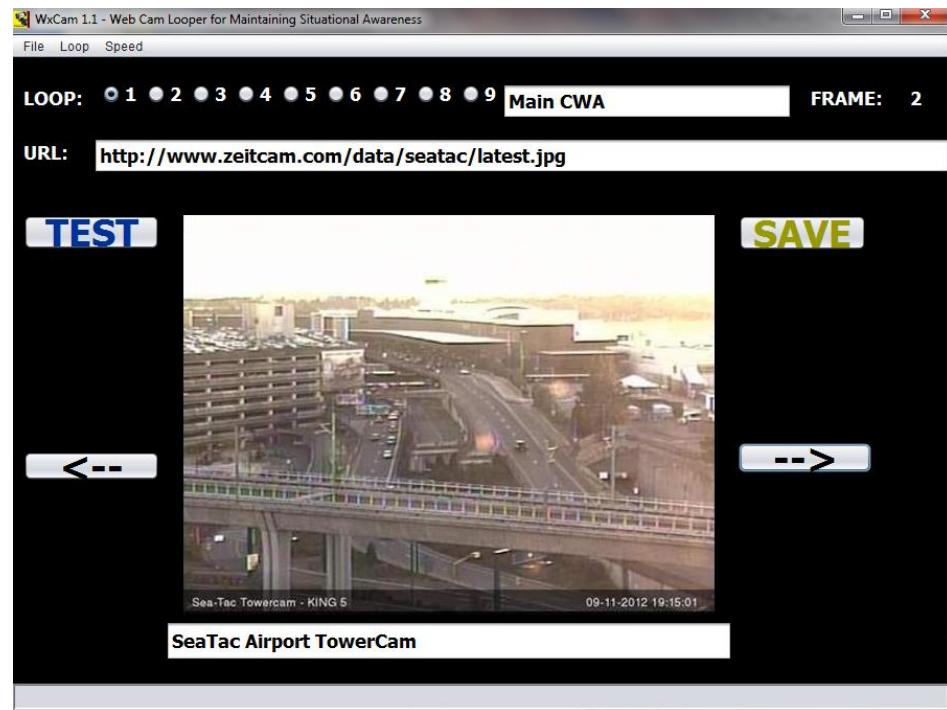


FIGURE 13. FRAME 2 OF LOOP 1 IS DISPLAYED AFTER CLICKING THE RIGHT ARROW BUTTON. NOTICE THAT THE LEFT ARROW BUTTON IS NOW ENABLED, ALLOWING YOU TO BACK UP TO THE PREVIOUS IMAGE. NOTE THAT THE DISPLAY IS SHOWN ONLY AFTER THE “TEST” BUTTON IS PRESSED.



FIGURE 14. RESULTS WHEN CLICKING LOOP RADIO BUTTON 5 IN NWS SEATTLE’S CONFIGURATION FILE AND CLICKING “TEST”.

Running WxCam:

WxCam allows you to change loops on the fly, change looping speed, pause loops, single step through loops, and resume looping. Loop management can be performed using mouse selections from the program menu or by using a keyboard's NUMPAD as keyboard accelerators (see Figure 15 and Figure 16). To use the NUMPAD, you'll need to have the "num lock" key toggled in the on position. When you start the looping process, images will change every 5 seconds by default. Use the "+" and "-" keys on the NUMPAD to speed or slow the loop speed by ½ second intervals with each press. The fastest loop speed possible is 1 frame per second, while the slowest loop speed possible is 1 frame per 30 seconds. Click the "." key on the NUMPAD to pause the loop, "*" to single step forward, "/" to single step backward, and "enter" to resume the looping process.

Use File->Close or CTRL-X to exit the application.

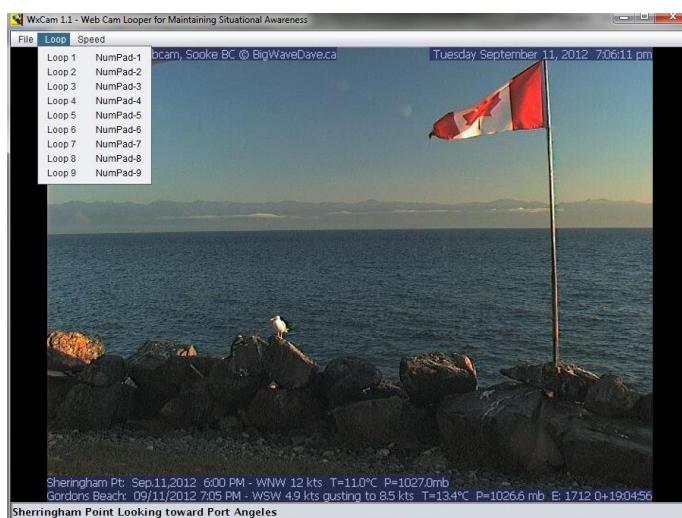


FIGURE 15. THE LOOP MENU WITH KEYBOARD ACCELERATORS.

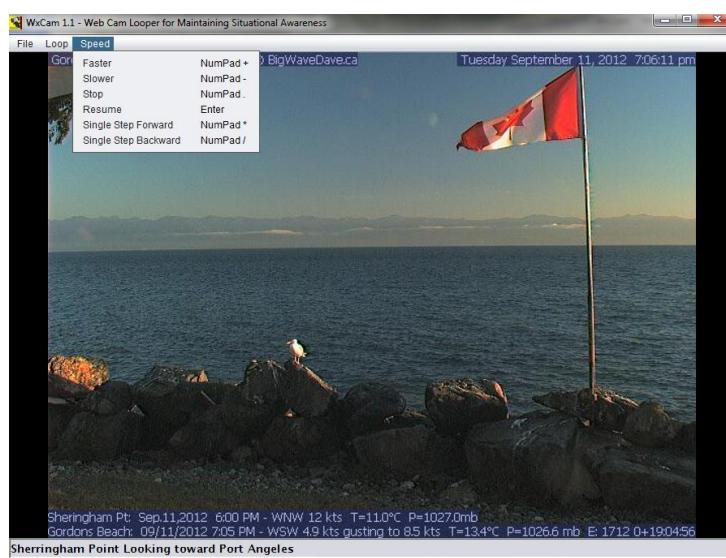


FIGURE 16. THE SPEED MENU WITH KEYBOARD ACCELERATORS.

Uninstalling WxCam:

To uninstall WxCam 1.3, open the Windows Control Panel and select WxCam as the program to uninstall (Figure 17). Then click “Yes” when you are asked if you are sure (Figure 18).

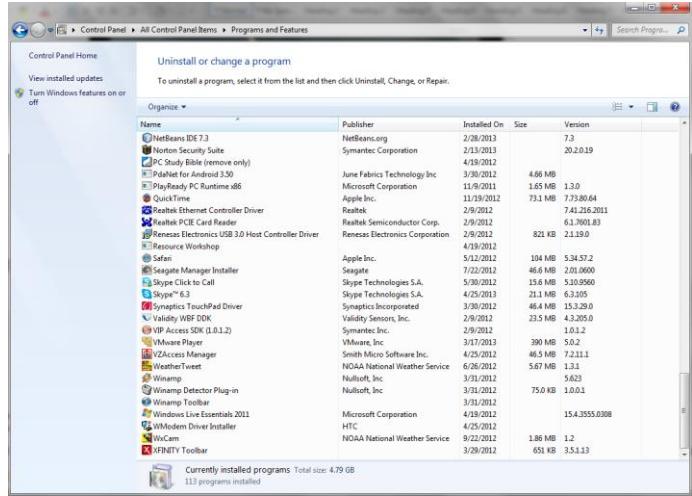


FIGURE 17. SELECT WxCAM FROM THE WINDOWS CONTROL PANEL UNINSTALL OR CHANGE A PROGRAM MENU.

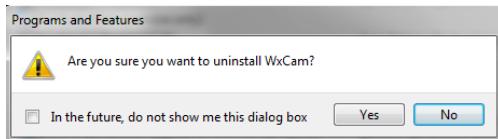


FIGURE 18. CLICK “YES” IF YOU ARE SURE YOU’D LIKE TO UNINSTALL THE APPLICATION.

The program’s directory and old imagery files will still be located in the WxCam directory (where you originally installed it – “Program Files (x86) -NOAA National Weather Service – WxCam” by default (see Figure 19.) You can manually delete the WxCam directory now from your system.

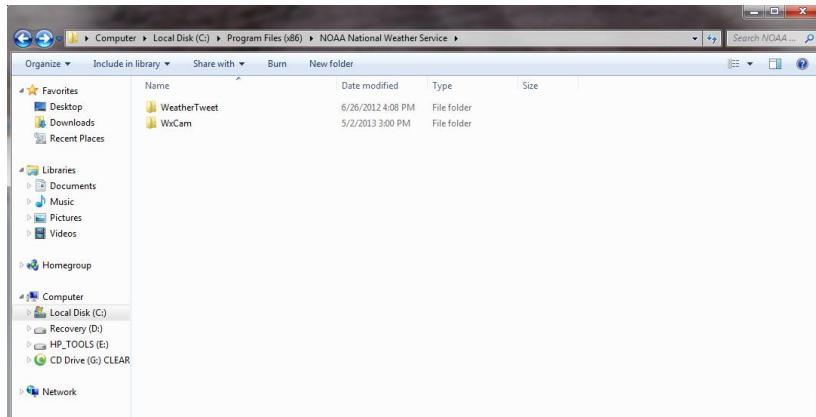


FIGURE 19. YOU CAN REMOVE WxCAM DIRECORY ONCE YOU HAVE COMPLTED THE UNINSTALL PROCESS. THIS WILL REMOVE THE DIRECTORY AND ANY OLD CACHE FILES IN THAT DIRECTORY.

Program Code:

WxCam 1.3 was written in Java using the Netbeans 7.2 Interactive Development Environment (free download from www.netbeans.org). Source code follows.

MainPanel.java:

```
package WxCam;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import javax.imageio.ImageIO;

/**
 * WxCam version 1.1. Adds the capability to single step forward and backward,
 * and adds the capability to resize the display window to fit any display or
 * to run as a postage stamp sized window.
 *
 * WxCam version 1.2 adds the capability to restart the background camera fetch
 * process if it stops unexpectedly.
 *
 * WxCam version 1.3 forces the click of TEST on the configuration tool to display
 * an image...this is to keep the program from hanging in the configuration process.
 *
 * @author Jay Albrecht, National Weather Service Seattle, May, 2013.
 */

public class MainPanel extends javax.swing.JFrame {

    // Use a series global of BufferedImage objects that hold images that
    // will be painted in various display panels. CurrentLoop and CurrentFrame
    // are used by the configuration portion of the GUI to keep track of the
    // frame and loop definitions held in WorkingConfig, a WxCamConfig object
    // that holds the URL and name of camera frames to be retrieved and loop
    // names. The program is designed to manage nine loops, each which can hold
    // up to 25 camera definitions.
}
```

```

// Member variables:

private BufferedImage MyImage;           // Image to be painted into Config
                                         // panel

private BufferedImage RetrievedImage;    // Working image retrieved for
                                         // Config purposes

private BufferedImage DRetrievedImage;   // Working image for live display
                                         // purposes

private BufferedImage BlankImage;        // Blank image for configuration

private BufferedImage DBlankImage;       // Blank image for live display

private FetchImagery GIMG = new FetchImagery(); // Image fetching thread

private javax.swing.Timer myTimer;        // Timer for looping purposes

private int LoopInterval = 5000;          // Initial loop interval (5 sec)

private WxCamConfig WorkingConfig;       // Working configuration object

private WxCamConfig LatestConfig;        // Latest saved configuration obj

private int CurrentLoop = 0;             // Current loop number (0-8)
                                         // for configuration panel

private int CurrentFrame = 0;            // Current frame number (0-24)
                                         // for configuration panel

private int CurrentDispLoop = 0;         // Current loop number (0-8)
                                         // for display panel

private int CurrentDispFrame = 0;        // Current frame number (0-24)
                                         // for display panel

/**
 * Creates new form NewJFrame
 */
public MainPanel() {
    initComponents();
}

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

    bgLoopConfigure = new javax.swing.ButtonGroup();
    StatusPanel = new javax.swing.JPanel();
    StatusLabel = new javax.swing.JLabel();
    FramePanel = new javax.swing.JPanel();
    ConfigurePanel = new javax.swing.JPanel();
    lblLoop = new javax.swing.JLabel();
    rbLoop1 = new javax.swing.JRadioButton();
    rbLoop2 = new javax.swing.JRadioButton();
    rbLoop3 = new javax.swing.JRadioButton();
    rbLoop4 = new javax.swing.JRadioButton();
    rbLoop5 = new javax.swing.JRadioButton();
    rbLoop6 = new javax.swing.JRadioButton();
    rbLoop7 = new javax.swing.JRadioButton();
    rbLoop8 = new javax.swing.JRadioButton();
    rbLoop9 = new javax.swing.JRadioButton();
    lblFrame = new javax.swing.JLabel();
    lblFrameNum = new javax.swing.JLabel();
    lblURL = new javax.swing.JLabel();
    edURL = new javax.swing.JTextField();
    pnTestCam = new javax.swing.JPanel();
}

```

```

        public void paint(Graphics gTestCam) {
            PaintTestCam(gTestCam);
        }
    };

    lb1FrameName = new javax.swing.JTextField();
    btnPrevImg = new javax.swing.JButton();
    btnNextImg = new javax.swing.JButton();
    btnTest = new javax.swing.JButton();
    btnSave = new javax.swing.JButton();
    edLoopName = new javax.swing.JTextField();
    CameraPanel = new javax.swing.JPanel();
    pn1Cam = new javax.swing.JPanel() {
        public void paint(Graphics g) {
            PaintCam(g);
        }
    };

    jMenuBar2 = new javax.swing.JMenuBar();
    mnuFile = new javax.swing.JMenu();
    mnuConfigure = new javax.swing.JMenuItem();
    jSeparator2 = new javax.swing.JPopupMenu.Separator();
    mnuLooper = new javax.swing.JMenuItem();
    jSeparator3 = new javax.swing.JPopupMenu.Separator();
    mnuReset = new javax.swing.JMenuItem();
    mnuClose = new javax.swing.JMenuItem();
    mnuLoopSelect = new javax.swing.JMenu();
    mnuLoop1 = new javax.swing.JMenuItem();
    mnuLoop2 = new javax.swing.JMenuItem();
    mnuLoop3 = new javax.swing.JMenuItem();
    mnuLoop4 = new javax.swing.JMenuItem();
    mnuLoop5 = new javax.swing.JMenuItem();
    mnuLoop6 = new javax.swing.JMenuItem();
    mnuLoop7 = new javax.swing.JMenuItem();
    mnuLoop8 = new javax.swing.JMenuItem();
    mnuLoop9 = new javax.swing.JMenuItem();
    mnuLoopSpeed = new javax.swing.JMenu();
    mnuFaster = new javax.swing.JMenuItem();
    mnuSlower = new javax.swing.JMenuItem();
    mnuStop = new javax.swing.JMenuItem();
    mnuResume = new javax.swing.JMenuItem();
    mnuStepFwd = new javax.swing.JMenuItem();
    mnuStepBwd = new javax.swing.JMenuItem();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    setTitle("WxCam 1.3 - Web Cam Looper for Maintaining Situational Awareness");
    setBackground(new java.awt.Color(0, 0, 0));
    setIconImage(Toolkit.getDefaultToolkit()
        .getImage("WxCam.png"));
    setMaximumSize(new java.awt.Dimension(2000, 1500));
    setMinimumSize(new java.awt.Dimension(100, 100));
    setName("frameCamera"); // NOI18N
    setPreferredSize(new java.awt.Dimension(1280, 720));

    StatusPanel.setBorder(javax.swing.BorderFactory.createBevelBorder(
        javax.swing.border.BevelBorder.LOWERED));
    StatusPanel.setMaximumSize(new java.awt.Dimension(2000, 25));
    StatusPanel.setMinimumSize(new java.awt.Dimension(100, 25));
    StatusPanel.setPreferredSize(new java.awt.Dimension(1280, 25));
    StatusPanel.setLayout(new java.awt.BorderLayout());

    StatusLabel.setFont(new java.awt.Font("Lucida Sans", 1, 14)); // NOI18N
    StatusLabel.setDoubleBuffered(true);
    StatusLabel.setMaximumSize(new java.awt.Dimension(1000, 14));
    StatusLabel.setMinimumSize(new java.awt.Dimension(100, 14));
    StatusLabel.setPreferredSize(new java.awt.Dimension(1000, 14));
    StatusPanel.add(StatusLabel, java.awt.BorderLayout.LINE_START);

    getContentPane().add(StatusPanel, java.awt.BorderLayout.PAGE_END);

    FramePanel.setBackground(new java.awt.Color(0, 0, 0));
    FramePanel.setForeground(new java.awt.Color(204, 255, 255));
    FramePanel.setMaximumSize(new java.awt.Dimension(2000, 1500));
    FramePanel.setMinimumSize(new java.awt.Dimension(100, 100));
    FramePanel.setLayout(new java.awt.BorderLayout());

    ConfigurePanel.setBackground(new java.awt.Color(0, 0, 0));
    ConfigurePanel.setForeground(new java.awt.Color(255, 255, 255));

```

```

ConfigurePanel.setMaximumSize(new java.awt.Dimension(2000, 1500));
ConfigurePanel.setMinimumSize(new java.awt.Dimension(100, 100));

lblLoop.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
lblLoop.setForeground(new java.awt.Color(255, 255, 255));
lblLoop.setText("LOOP:");

rbLoop1.setBackground(new java.awt.Color(102, 51, 0));
bgLoopConfigure.add(rbLoop1);
rbLoop1.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
rbLoop1.setForeground(new java.awt.Color(255, 255, 255));
rbLoop1.setSelected(true);
rbLoop1.setText("1");
rbLoop1.setDoubleBuffered(true);
rbLoop1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        LoopConfigHandler(evt);
    }
});

rbLoop2.setBackground(new java.awt.Color(102, 51, 0));
bgLoopConfigure.add(rbLoop2);
rbLoop2.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
rbLoop2.setForeground(new java.awt.Color(255, 255, 255));
rbLoop2.setText("2");
rbLoop2.setDoubleBuffered(true);
rbLoop2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        LoopConfigHandler(evt);
    }
});

rbLoop3.setBackground(new java.awt.Color(102, 51, 0));
bgLoopConfigure.add(rbLoop3);
rbLoop3.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
rbLoop3.setForeground(new java.awt.Color(255, 255, 255));
rbLoop3.setText("3");
rbLoop3.setDoubleBuffered(true);
rbLoop3.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        LoopConfigHandler(evt);
    }
});

rbLoop4.setBackground(new java.awt.Color(102, 51, 0));
bgLoopConfigure.add(rbLoop4);
rbLoop4.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
rbLoop4.setForeground(new java.awt.Color(255, 255, 255));
rbLoop4.setText("4");
rbLoop4.setDoubleBuffered(true);
rbLoop4.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        LoopConfigHandler(evt);
    }
});

rbLoop5.setBackground(new java.awt.Color(102, 51, 0));
bgLoopConfigure.add(rbLoop5);
rbLoop5.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
rbLoop5.setForeground(new java.awt.Color(255, 255, 255));
rbLoop5.setText("5");
rbLoop5.setDoubleBuffered(true);
rbLoop5.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        LoopConfigHandler(evt);
    }
});

rbLoop6.setBackground(new java.awt.Color(102, 51, 0));
bgLoopConfigure.add(rbLoop6);
rbLoop6.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
rbLoop6.setForeground(new java.awt.Color(255, 255, 255));
rbLoop6.setText("6");
rbLoop6.setDoubleBuffered(true);
rbLoop6.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        LoopConfigHandler(evt);
    }
});

```

```

});

rbLoop7.setBackground(new java.awt.Color(102, 51, 0));
bgLoopConfigure.add(rbLoop7);
rbLoop7.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
rbLoop7.setForeground(new java.awt.Color(255, 255, 255));
rbLoop7.setText("7");
rbLoop7.setDoubleBuffered(true);
rbLoop7.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        LoopConfigHandler(evt);
    }
});

rbLoop8.setBackground(new java.awt.Color(102, 51, 0));
bgLoopConfigure.add(rbLoop8);
rbLoop8.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
rbLoop8.setForeground(new java.awt.Color(255, 255, 255));
rbLoop8.setText("8");
rbLoop8.setDoubleBuffered(true);
rbLoop8.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        LoopConfigHandler(evt);
    }
});

rbLoop9.setBackground(new java.awt.Color(102, 51, 0));
bgLoopConfigure.add(rbLoop9);
rbLoop9.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
rbLoop9.setForeground(new java.awt.Color(255, 255, 255));
rbLoop9.setText("9");
rbLoop9.setDoubleBuffered(true);
rbLoop9.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        LoopConfigHandler(evt);
    }
});

lblFrame.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
lblFrame.setForeground(new java.awt.Color(255, 255, 255));
lblFrame.setText("FRAME:");
lblFrame.setDoubleBuffered(true);

lblFrameNum.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
lblFrameNum.setForeground(new java.awt.Color(255, 255, 255));
lblFrameNum.setHorizontalTextPosition(javax.swing.SwingConstants.CENTER);
lblFrameNum.setText("1");
lblFrameNum.setDoubleBuffered(true);
lblFrameNum.setMaximumSize(new java.awt.Dimension(50, 22));
lblFrameNum.setMinimumSize(new java.awt.Dimension(50, 22));
lblFrameNum.setPreferredSize(new java.awt.Dimension(50, 22));

lblURL.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
lblURL.setForeground(new java.awt.Color(255, 255, 255));
lblURL.setText("URL: ");
lblURL.setVerticalTextPosition(javax.swing.SwingConstants.BOTTOM);
lblURL.setDoubleBuffered(true);

edURL.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
edURL.setDoubleBuffered(true);

pn1TestCam.setBackground(new java.awt.Color(0, 0, 0));
pn1TestCam.setMinimumSize(new java.awt.Dimension(100, 100));

org.jdesktop.layout.GroupLayout pn1TestCamLayout = new org.jdesktop.layout.GroupLayout(pn1TestCam);
pn1TestCam.setLayout(pn1TestCamLayout);
pn1TestCamLayout.setHorizontalGroup(
    pn1TestCamLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
        .add(0, 542, Short.MAX_VALUE)
);
pn1TestCamLayout.setVerticalGroup(
    pn1TestCamLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
        .add(0, 382, Short.MAX_VALUE)
);

lblFrameName.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
lblFrameName.setDoubleBuffered(true);

```

```

btnPrevImg.setFont(new java.awt.Font("Tahoma", 1, 36)); // NOI18N
btnPrevImg.setText("<--");
btnPrevImg.setDoubleBuffered(true);
btnPrevImg.setEnabled(false);
btnPrevImg.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        PrevFrame(evt);
    }
});
btnNextImg.setFont(new java.awt.Font("Tahoma", 1, 36)); // NOI18N
btnNextImg.setText("-->");
btnNextImg.setDoubleBuffered(true);
btnNextImg.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        NextFrame(evt);
    }
});
btnTest.setFont(new java.awt.Font("Tahoma", 1, 36)); // NOI18N
btnTest.setForeground(new java.awt.Color(0, 51, 153));
btnTest.setText("TEST");
btnTest.setDoubleBuffered(true);
btnTest.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        TestImage(evt);
    }
});
btnSave.setFont(new java.awt.Font("Tahoma", 1, 36)); // NOI18N
btnSave.setForeground(new java.awt.Color(153, 153, 0));
btnSave.setText("SAVE");
btnSave.setDoubleBuffered(true);
btnSave.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        SaveConfig(evt);
    }
});
edLoopName.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
edLoopName.setDoubleBuffered(true);

org.jdesktop.layout.GroupLayout ConfigurePanelLayout = new
    org.jdesktop.layout.GroupLayout(ConfigurePanel);
ConfigurePanel.setLayout(ConfigurePanelLayout);
ConfigurePanelLayout.setHorizontalGroup(
    ConfigurePanelLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
        .add(ConfigurePanelLayout.createSequentialGroup()
            .add(10, 10, 10)
            .add(lblLoop)
            .add(18, 18, 18)
            .add(rbLoop1)
            .add(10, 10, 10)
            .add(rbLoop2)
            .add(10, 10, 10)
            .add(rbLoop3)
            .add(10, 10, 10)
            .add(rbLoop4)
            .add(10, 10, 10)
            .add(rbLoop5)
            .add(10, 10, 10)
            .add(rbLoop6)
            .add(10, 10, 10)
            .add(rbLoop7)
            .add(10, 10, 10)
            .add(rbLoop8)
            .add(10, 10, 10)
            .add(rbLoop9)
            .add(6, 6, 6)
            .add(edLoopName, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 277,
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
            .add(18, 18, 18)
            .add(lblFrame)
            .add(6, 6, 6)
            .add(lblFrameNum, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
                org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
            .add(ConfigurePanelLayout.createSequentialGroup()

```

```

        .add(10, 10, 10)
        .add(lblURL, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 57,
              org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
        .add(10, 10, 10)
        .add(edURL, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 1193,
              org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
    add(ConfigurePanelLayout.createSequentialGroup()
        .add(10, 10, 10)
        .add(ConfigurePanelLayout.createParallelGroup(
            org.jdesktop.layout.GroupLayout.LEADING)
            .add(btnTest, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 130,
                  org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
            .add(btnPrevImg, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 130,
                  org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
        .add(6, 6, 6)
        .add(pnlTestCam, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
              org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
              org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
        .add(6, 6, 6)
        .add(ConfigurePanelLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(btnSave)
            .add(btnNextImg, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 127,
                  org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
    add(ConfigurePanelLayout.createSequentialGroup()
        .add(146, 146, 146)
        .add(lblFrameName, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 542,
              org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
);
ConfigurePanelLayout.setVerticalGroup(
    ConfigurePanelLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
        .add(ConfigurePanelLayout.createSequentialGroup()
            .add(24, 24, 24)
            .add(ConfigurePanelLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
                .add(ConfigurePanelLayout.createSequentialGroup()
                    .add(4, 4, 4)
                    .add(lblLoop))
                .add(rbLoop1)
                .add(rbLoop2)
                .add(rbLoop3)
                .add(rbLoop4)
                .add(rbLoop5)
                .add(rbLoop6)
                .add(rbLoop7)
                .add(rbLoop8)
                .add(rbLoop9)
                .add(ConfigurePanelLayout.createSequentialGroup()
                    .add(1, 1, 1)
                    .add(edLoopName, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
                          org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                          org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
            .add(ConfigurePanelLayout.createSequentialGroup()
                .add(4, 4, 4)
                .add(lblFrame))
            .add(ConfigurePanelLayout.createSequentialGroup()
                .add(4, 4, 4)
                .add(lblFrameNum, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
                      org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                      org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
        .add(18, 18, 18)
        .add(ConfigurePanelLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(ConfigurePanelLayout.createSequentialGroup()
                .add(3, 3, 3)
                .add(lblURL))
            .add(edURL, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
                  org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                  org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
        .add(40, 40, 40)
        .add(ConfigurePanelLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
            .add(ConfigurePanelLayout.createSequentialGroup()
                .add(btnTest, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 33,
                      org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
                .add(193, 193, 193)
                .add(btnPrevImg, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 28,
                      org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
            .add(pnlTestCam, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
                  org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,
                  org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
        .add(ConfigurePanelLayout.createSequentialGroup()

```

```

        .add(btnSave, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 34,
        org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
        .add(184, 184, 184)
        .add(btnNextImg, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 28,
        org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))
    .add(6, 6)
    .add(lblFrameName, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE, 38,
        org.jdesktop.layout.GroupLayout.PREFERRED_SIZE))
);

FramePanel.add(ConfigurePanel, java.awt.BorderLayout.CENTER);
ConfigurePanel.setVisible(false);
SetConfigPanel();

CameraPanel.setBackground(new java.awt.Color(0, 0, 0));
CameraPanel.setForeground(new java.awt.Color(255, 255, 255));
CameraPanel.setMaximumSize(new java.awt.Dimension(2000, 1500));
CameraPanel.setMinimumSize(new java.awt.Dimension(100, 100));

pnlCam.setBackground(new java.awt.Color(0, 0, 0));
pnlCam.setMaximumSize(new java.awt.Dimension(2000, 1500));
pnlCam.setMinimumSize(new java.awt.Dimension(100, 100));
pnlCam.setPreferredSize(new java.awt.Dimension(1260, 733));

org.jdesktop.layout.GroupLayout pnlCamLayout = new org.jdesktop.layout.GroupLayout(pnlCam);
pnlCam.setLayout(pnlCamLayout);
pnlCamLayout.setHorizontalGroup(
    pnlCamLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
        .addGroup(0, 1260, Short.MAX_VALUE)
);
pnlCamLayout.setVerticalGroup(
    pnlCamLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
        .addGroup(0, 733, Short.MAX_VALUE)
);

org.jdesktop.layout.GroupLayout CameraPanelLayout = new org.jdesktop.layout.GroupLayout(CameraPanel);
CameraPanel.setLayout(CameraPanelLayout);
CameraPanelLayout.setHorizontalGroup(
    CameraPanelLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
        .addGroup(pnlCam, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
            org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
);
CameraPanelLayout.setVerticalGroup(
    CameraPanelLayout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)
        .addGroup(pnlCam, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,
            org.jdesktop.layout.GroupLayout.DEFAULT_SIZE, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)
);

FramePanel.add(CameraPanel, java.awt.BorderLayout.PAGE_START);

getContentPane().add(FramePanel, java.awt.BorderLayout.CENTER);
StartImageFetch();

jMenuBar2.setDoubleBuffered(true);
jMenuBar2.setMaximumSize(new java.awt.Dimension(2000, 21));
jMenuBar2.setMinimumSize(new java.awt.Dimension(100, 21));
jMenuBar2.setPreferredSize(new java.awt.Dimension(1280, 21));

mnuFile.setText("File");
mnuFile.setDoubleBuffered(true);

mnuConfigure.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_C,
    java.awt.event.InputEvent.CTRL_MASK));
mnuConfigure.setText("Configure Loops");
mnuConfigure.setActionCommand("Configure");
mnuConfigure.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        MenuHandler(evt);
    }
});
mnuFile.add(mnuConfigure);
mnuFile.add(jSeparator2);

mnuLooper.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_D,
    java.awt.event.InputEvent.CTRL_MASK));
mnuLooper.setText("Display Camera Loops");
mnuLooper.setActionCommand("Display");
mnuLooper.addActionListener(new java.awt.event.ActionListener() {

```

```

        public void actionPerformed(java.awt.event.ActionEvent evt) {
            MenuHandler(evt);
        }
    });
    mnuFile.add(mnuLooper);
    mnuFile.add(jSeparator3);

    mnuReset.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_R,
        java.awt.event.InputEvent.CTRL_MASK));
    mnuReset.setText("Reset Camera Fetch Process");
    mnuReset.setActionCommand("Reset");
    mnuReset.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            MenuHandler(evt);
        }
    });
    mnuFile.add(mnuReset);

    mnuClose.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_X,
        java.awt.event.InputEvent.CTRL_MASK));
    mnuClose.setText("Close");
    mnuClose.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            MenuHandler(evt);
        }
    });
    mnuFile.add(mnuClose);

    jMenuBar2.add(mnuFile);

    mnuLoopSelect.setText(" Loop");
    mnuLoopSelect.setDoubleBuffered(true);

    mnuLoop1.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_NUMPAD1, 0));
    mnuLoop1.setText("Loop 1");
    mnuLoop1.setActionCommand("1");
    mnuLoop1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            LoopHandler(evt);
        }
    });
    mnuLoopSelect.add(mnuLoop1);

    mnuLoop2.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_NUMPAD2, 0));
    mnuLoop2.setText("Loop 2");
    mnuLoop2.setActionCommand("2");
    mnuLoop2.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            LoopHandler(evt);
        }
    });
    mnuLoopSelect.add(mnuLoop2);

    mnuLoop3.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_NUMPAD3, 0));
    mnuLoop3.setText("Loop 3");
    mnuLoop3.setActionCommand("3");
    mnuLoop3.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            LoopHandler(evt);
        }
    });
    mnuLoopSelect.add(mnuLoop3);

    mnuLoop4.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_NUMPAD4, 0));
    mnuLoop4.setText("Loop 4");
    mnuLoop4.setActionCommand("4");
    mnuLoop4.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            LoopHandler(evt);
        }
    });
    mnuLoopSelect.add(mnuLoop4);

    mnuLoop5.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_NUMPAD5, 0));
    mnuLoop5.setText("Loop 5");
    mnuLoop5.setActionCommand("5");
    mnuLoop5.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {

```

```

        LoopHandler(evt);
    }
});
mnuLoopSelect.add(mnuLoop5);

mnuLoop6.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_NUMPAD6, 0));
mnuLoop6.setText("Loop 6");
mnuLoop6.setActionCommand("6");
mnuLoop6.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        LoopHandler(evt);
    }
});
mnuLoopSelect.add(mnuLoop6);

mnuLoop7.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_NUMPAD7, 0));
mnuLoop7.setText("Loop 7");
mnuLoop7.setActionCommand("7");
mnuLoop7.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        LoopHandler(evt);
    }
});
mnuLoopSelect.add(mnuLoop7);

mnuLoop8.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_NUMPAD8, 0));
mnuLoop8.setText("Loop 8");
mnuLoop8.setActionCommand("8");
mnuLoop8.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        LoopHandler(evt);
    }
});
mnuLoopSelect.add(mnuLoop8);

mnuLoop9.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_NUMPAD9, 0));
mnuLoop9.setText("Loop 9");
mnuLoop9.setActionCommand("9");
mnuLoop9.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        LoopHandler(evt);
    }
});
mnuLoopSelect.add(mnuLoop9);

jMenuBar2.add(mnuLoopSelect);

mnuLoopSpeed.setText(" Speed");
mnuLoopSpeed.setDoubleBuffered(true);

mnuFaster.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_ADD, 0));
mnuFaster.setText("Faster");
mnuFaster.setActionCommand("FAST");
mnuFaster.setDoubleBuffered(true);
mnuFaster.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        SpeedHandler(evt);
    }
});
mnuLoopSpeed.add(mnuFaster);

mnuSlower.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_SUBTRACT, 0));
mnuSlower.setText("Slower");
mnuSlower.setActionCommand("SLOW");
mnuSlower.setDoubleBuffered(true);
mnuSlower.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        SpeedHandler(evt);
    }
});
mnuLoopSpeed.add(mnuSlower);

mnuStop.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_DECIMAL, 0));
mnuStop.setText("Stop");
mnuStop.setActionCommand("STOP");
mnuStop.setDoubleBuffered(true);
mnuStop.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {

```

```

        SpeedHandler(evt);
    });
mnuLoopSpeed.add(mnuStop);

mnuResume.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_ENTER, 0));
mnuResume.setText("Resume");
mnuResume.setActionCommand("RESUME");
mnuResume.setDoubleBuffered(true);
mnuResume.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        SpeedHandler(evt);
    }
});
mnuLoopSpeed.add(mnuResume);

mnuStepFwd.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_MULTIPLY, 0));
mnuStepFwd.setText("Single Step Forward");
mnuStepFwd.setActionCommand("FWD");
mnuStepFwd.setDoubleBuffered(true);
mnuStepFwd.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        SpeedHandler(evt);
    }
});
mnuLoopSpeed.add(mnuStepFwd);

mnuStepBwd.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_DIVIDE, 0));
mnuStepBwd.setText("Single Step Backward");
mnuStepBwd.setActionCommand("BWD");
mnuStepBwd.setDoubleBuffered(true);
mnuStepBwd.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        SpeedHandler(evt);
    }
});
mnuLoopSpeed.add(mnuStepBwd);

jMenuBar2.add(mnuLoopSpeed);

setJMenuBar(jMenuBar2);

getAccessibleContext().setAccessibleName("WxCam 1.3");

pack();
}// </editor-fold>

private void MenuHandler(java.awt.event.ActionEvent evt) {

    // Handle the clicking of any of the menu items. They all activate
    // the Action Performed event. There are three options presented to the
    // user: Configure (opens the configuration GUI), Display: shows the
    // frames on a full screen background in a looping manner, and Close:
    // exits the application.

    switch (evt.getActionCommand()) {

        case "Close":
            if (GIMG != null) {
                StopImageFetch();
            }
            if (myTimer != null) {
                myTimer.stop();
                myTimer = null;
            }
            System.exit(0);
            break;
        case "Configure":

```

```

        if (GIMG != null) {
            StopImageFetch();
        }

        if (myTimer != null) {
            myTimer.stop();
        }

        SetConfigPanel();
        StatusLabel.setText("  Setting Configuration of Camera Loops");
        CameraPanel.setVisible(false);
        ConfigurePanel.setVisible(true);

        break;

    case "Display":
        if (GIMG != null) {
            StopImageFetch();
        }

        StartImageFetch();
        StatusLabel.setText("");
        ConfigurePanel.setVisible(false);
        CameraPanel.setVisible(true);
        pnlCam.setVisible(true);
        DisplayMainGraphics();

        break;

    case "Reset":
        if (GIMG != null) {
            StopImageFetch();
        }
        StartImageFetch();
        break;
    }

}

// Start fetching imagery in the background. Do this by stoping then
// destroying the running thread (if there), then creating and starting
// a new thread.

private void StartImageFetch() {

    if (GIMG != null) {
        StopImageFetch();
    }

    GIMG = new FetchImagery();
    GIMG.start();
}

// Stop fetching imagery in the background then destroy the fetch imagery
// thread object.

private void StopImageFetch() {
    GIMG.endThread();
    GIMG = null;
}

```

```

// LOOPER responds to a firing of the timer (which can fire off between 1
// and 30 second intervals with a default value of 5 seconds). It reads
// the latest WxCamConfig file. It then uses CurrentDispLoop and
// CurrentDispFrame to determine which image to display. The
// value of CurrentDispFrame is cycled with each firing of the timer, if the
// value is greater than 24 after cycling, it is reset to 0.

// If the loop is empty, a message is displayed on the status panel. If an
// individual frame is empty, the counter cycles until it isn't.

// DRetrievedImage, when changed, will cause the panel that holds the
// image to repaint with its current value.

ActionListener LOOPER = new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent AE) {

        LatestConfig = WxCamConfig.getConfigFile();

        // Create a blank black image (to be used if a real one is
        // unavailable).

        DBlankImage =
            new BufferedImage(2000, 1500, BufferedImage.TYPE_INT_RGB);
        Graphics g = DBlankImage.getGraphics();
        g.setColor(Color.BLACK);
        g.fillRect(0, 0, 2000, 1500);
        g.dispose();

        // If the loop name of the currently selected loop is empty, the
        // loop is defined as empty. A message is displayed in the
        // status panel and no further processing is performed.

        if (LatestConfig.Loops[CurrentDispLoop].LoopName.
            trim().length() < 1) {

            StatusLabel.setText("THIS LOOP IS EMPTY!");

        }

        else {

            // Continue processing since the loop is not empty.
            // Calculate a filename L#F##.CMR based on the current values
            // of CurrentDispLoop and CurrentDispFrame.

            String FN = "L" + Integer.toString(CurrentDispLoop) + "F" +
                Integer.toString(CurrentDispFrame) + ".CMR";

            // Check to see if the file containing the requested image is
            // available.

            Boolean avail = ImageAvailable(FN);

            // If not available, step through the loop until an image is
            // found.

            while (!avail) {

                CurrentDispFrame++;

                if( CurrentDispFrame > 24) {

                    CurrentDispFrame = 0;
                }

                FN = "L" + Integer.toString(CurrentDispLoop) + "F" +
                    Integer.toString(CurrentDispFrame) + ".CMR";
                avail = ImageAvailable(FN);

            }

            // Once an image is found to display, read the file, set
            // the image into DRetrievedImage, then display the name of
            // the camera into the status panel.

            try {

```

```

        DRetrievedImage = ImageIO.read(new File(FN));
    }
    catch (Throwable t) { }

    StatusLabel.setText(LatestConfig.Loops[CurrentDispLoop].
        Cameras[CurrentDispFrame].imageName);

    // Go to the next frame in the loop when done. This will be
    // the starting point the next time the timer is fired.
    // The CurrentDispFrame will lie between 0 and 24 inclusive.

    CurrentDispFrame++;
    if (CurrentDispFrame > 24) {
        CurrentDispFrame = 0;
    }
}

};

// This method is called when the user selects to display looping images.
// A new timer object is created with a default firing of 5 seconds, then
// the actionlistener is assigned to LOOPER. The timer is then started.

private void DisplayMainGraphics() {

    myTimer = new javax.swing.Timer(5000, LOOPER);
    myTimer.start();
}

// ImageAvailable(String ImageName) looks in the directory the application
// is located and searches for the file ImageName. If the image is found
// in the directory, TRUE is returned and if not, FALSE is returned.

private Boolean ImageAvailable(String ImageName) {

    Boolean isAvailable;

    File file=new File(ImageName);
    isAvailable = file.exists();

    return isAvailable;
}

// SetConfigPanel is called when the configure panel is all set up upon
// start of the program. It creates a black blank image that can be used
// as a placeholder when there is nothing to display. The method also
// initializes the CurrentLoop and CurrentFrame values to zero and sets the
// btnPrevImg button to disabled as the frame number can only go down to
// zero. Finally, the method reads in the configuration file (if it can) and
// sets the panel up with initial values.

private void SetConfigPanel() {

    // Make a Black blank image

    BlankImage = new BufferedImage(678, 449, BufferedImage.TYPE_INT_RGB);
    Graphics g = BlankImage.getGraphics();
    g.setColor(Color.BLACK);
    g.fillRect(0, 0, BlankImage.getWidth(), BlankImage.getHeight());

    // Initialize variables and button states.

    WorkingConfig = null;
    CurrentLoop = 0;
    CurrentFrame = 0;
    btnPrevImg.setEnabled(false);
    btnNextImg.setEnabled(true);
}

```

```

// Read in the configuration file (it will be blank the first time run.
WorkingConfig = WxCamConfig.getConfigFile();

// Fill in the panel as appropriate.
RefreshConfigPanel();

}

// RefreshConfigPanel() displays information in the Configuration Panel
// based on the current values of CurrentLoop and CurrentFrame. If
// appropriate, the URL of the CurrentFrame, the name of the image, and
// a current picture are placed on the form.

private void RefreshConfigPanel() {

    // Place the name of CurrentLoop in the edLoopName Text Box.

    if (WorkingConfig.Loops[CurrentLoop].LoopName.trim().length() > 0) {
        edLoopName.setText(WorkingConfig.Loops[CurrentLoop].LoopName);
    }
    else {
        edLoopName.setText("");
    }

    // Place the CurrentFrame+1 in the lblFrameNum label field
    lblFrameNum.setText(Integer.toString(CurrentFrame + 1));

    // Place the URL of the CurrentFrame in the edURL edit box.

    edURL.setText(WorkingConfig.Loops[CurrentLoop].Cameras[CurrentFrame].
        imageURL.trim());

    // Place the Name of the CurrentFrame image in lblFrameName.

    lblFrameName.setText(WorkingConfig.Loops[CurrentLoop].
        Cameras[CurrentFrame].imageName);

    // Try to display the image at the selected URL in the TestCam frame.
    // If one is not available, the TestImage routine will just draw a black
    // empty frame into the frame.

    try {
        // TestImage();
    }
    catch (Throwable t) {}

}

// ResizeImage takes an original Buffered Image and resizes it to height H
// using a bicubic interpolation that maintains aspect ratio. The process
// is best for ensuring image quality during a resizing process.
// ResizeImage returns a new BufferedImage.

private BufferedImage ResizeImage(BufferedImage OriginalImage, int W, int H) {

    // Compute the width of the new image based on its original height and
    // the height you want it to be. This new height will normally be the
    // height of the panel that will display the image.

    double newW = (double)(OriginalImage.getWidth())*((double)H)/
        ((double)OriginalImage.getHeight());
    int NEWW = (int)newW;

    // Resize the image using bicubic interpolation.

    BufferedImage resizedImg = new BufferedImage(NEWW, H,
        OriginalImage.getType());

```

```

Graphics2D g = resizedImg.createGraphics();
g.setRenderingHint(RenderingHints.KEY_INTERPOLATION,
    RenderingHints.VALUE_INTERPOLATION_BICUBIC);
g.drawImage(OriginalImage, 0, 0, NEWW, H, 0, 0, OriginalImage.getWidth(),
    OriginalImage.getHeight(), null);
g.dispose();

// Return the resized image

return resizedImg;

}

// PaintTestCam(Graphics gTestCam) paints or repaints the global
// BufferedImage MyImage into the TestCam panel. If MyImage is null - or not
// available for some reason, the blank black image BlankImage is drawn onto
// the panel instead. The panel and precreated blank image are the same size.

private void PaintTestCam(Graphics gTestCam) {

    // Draw the Blank Black image into pnlTestCam

    gTestCam.drawImage(BlankImage, 0, 0, pnlTestCam);

    // Try to draw myImage into pnlTestCam as a centered image.

    if (MyImage != null) {

        int startX = (pnlTestCam.getWidth() - MyImage.getWidth())/2;
        gTestCam.drawImage(MyImage, startX, 0, pnlTestCam);

    }

}

// PaintCam(Graphics g) paints or repaints a resized version of the
// BufferedImage DRetrievedImage into pnlCam, the only panel on the
// applications Display panel.

private void PaintCam(Graphics g) {

    // get the height and width of the current application:

    int h = FramePanel.getHeight(); // was FramePanel
    int w = FramePanel.getWidth();
    CameraPanel.setSize(w, h);
    pnlCam.setSize(w, h);

    BufferedImage Resize;

    // Draw a blank black image into pnlCam.

    g.drawImage(DBlankImage, 0, 0, pnlCam);

    // Resize DRetrievedImage to fit in pnlCam, then draw the resized image
    // in centered position.

    if (DRetrievedImage != null) {

        Resize = ResizeImage(DRetrievedImage, w, h);

        //int imageWidth = Resize.getWidth();
        int startX = (w - Resize.getWidth())/2;

        g.drawImage(Resize, startX, 0, pnlCam);

    }

}

// TestImage() allows the user to test a URL entered on the configuration
// panel in order to determine if an image can be successfully retrieved and
// displayed by this program from the website specified in the URL edit box.
// This method is also run when navigating around the configuration section
// of the GUI.

```

```

// TestImage() attempts to create a URL object from the edit box,
// retrieve an image from the specified URL, then resizes the image to
// the size of the display panel and displays the image in that panel.
// If unsuccessful, a message is displayed in the status panel at the
// bottom of the GUI and a blank black image is displayed as a placeholder.

private void TestImage(){

    BufferedImage Resized;
    URL url = null;

    // Determine if the URL is malformed.

    try {

        url = new URL(edURL.getText().trim());
        StatusLabel.setText("");

    }

    catch (MalformedURLException e) {

        String msg = e.getMessage() + " Check your URL entry!";
        StatusLabel.setText(msg);

    }

    // First set the retrieved image to be a blank black image.

    RetrievedImage = new BufferedImage(BlankImage.getWidth(),
        BlankImage.getHeight(), BufferedImage.TYPE_INT_RGB);
    RetrievedImage.setData(BlankImage.getData());

    // Now try to download the image from the URL. If there is a problem
    // display a message on the status panel.

    try {

        RetrievedImage = ImageIO.read(url);
        Resized = ResizeImage(RetrievedImage,
            pnlTestCam.getWidth(), pnlTestCam.getHeight());
        MyImage = Resized;
        StatusLabel.setText("");

    }

    catch (IOException e) {

        String msg = e.getMessage();
        StatusLabel.setText(msg + " Problem retrieving image from" + url);

    }

    // Draw the image in the pnlTestCam

    pnlTestCam.repaint();

}

// If the button presses the TEST button on the configuration panel,
// read the URL in the URL edit box, attempt to go online and retrieve
// the image, resize the image to the size of the frame, then display
// the image in the test panel. This method is simply an event handler for
// the button click and calls an overloaded version of TestImage();

private void TestImage(java.awt.event.ActionEvent evt) {

    TestImage();

}

// PrevFrame responds to the click of the PrevFrame button on the
// configuration panel. When clicked, decrement CurrentFrame (unless it is
// zero), then use RefreshConfigPanel() to update the contents of the panel
// based on the values of CurrentLoop and CurrentFrame.

private void PrevFrame(java.awt.event.ActionEvent evt) {

```

```

// Blank out MyImage and repaint in order to erase the pnlTestCam panel.

MyImage = null;
pnlTestCam.repaint();

// decrement CurrentFrame with a minimum of 0.

CurrentFrame--;
if (CurrentFrame < 0) {
    CurrentFrame = 0;
}

// disable and enable btnPrevImg and btnNextImg as appropriate based
// on the value of CurrentFrame.

if (CurrentFrame == 0) {
    btnPrevImg.setEnabled(false);
} else {
    btnPrevImg.setEnabled(true);
}

if (CurrentFrame == 24) {
    btnNextImg.setEnabled(false);
} else {
    btnNextImg.setEnabled(true);
}

// Refresh the configuration panel.

RefreshConfigPanel();
}

// NextFrame responds to the click of the NextFrame button on the
// configuration panel. When clicked, increment CurrentFrame (unless it is
// 24), then use RefreshConfigPanel() to update the contents of the panel
// based on the values of CurrentLoop and CurrentFrame.

private void NextFrame(java.awt.event.ActionEvent evt) {
    // Blank out MyImage and repaint in order to erase the pnlTestCam panel.

    MyImage = null;
    pnlTestCam.repaint();

    // increment CurrentFrame with a maximum of 24

    CurrentFrame++;
    if (CurrentFrame > 24) {
        CurrentFrame = 24;
    }

    // disable and enable btnPrevImg and btnNextImg as appropriate based
    // on the value of CurrentFrame.

    if (CurrentFrame == 24) {
        btnNextImg.setEnabled(false);
    }
}

```

```

    else {
        btnNextImg.setEnabled(true);
    }

    if (CurrentFrame == 0) {
        btnPrevImg.setEnabled(false);
    }
    else {
        btnPrevImg.setEnabled(true);
    }

    // Refresh the configuration panel.
    RefreshConfigPanel();
}

// SaveConfig responds to the click of the SAVE button on the configuration
// panel. Save any changes to loop name, image URL, and image name that may
// have been made to the WorkingConfig object, then save the object to file.

private void SaveConfig(java.awt.event.ActionEvent evt) {

    WorkingConfig.Loops[CurrentLoop].Cameras[CurrentFrame].
        imageURL = edURL.getText();
    WorkingConfig.Loops[CurrentLoop].Cameras[CurrentFrame].
        imageName = lblFrameName.getText();
    WorkingConfig.Loops[CurrentLoop].LoopName = edLoopName.getText();

    WxCamConfig.saveConfigFile(WorkingConfig);
}

// LoopConfigHandler responds to the click of any of the various loop radio
// buttons on the configuration panel. Blank out myImage, repaint the
// TestCam panel to place a blank and black image on it, change to the
// indicated loop, set to frame zero, set btnPrevImg to disabled, then
// refresh.

private void LoopConfigHandler(java.awt.event.ActionEvent evt) {

    MyImage = null;
    pn1TestCam.repaint();
    CurrentLoop = Integer.parseInt(evt.getActionCommand()) - 1;
    CurrentFrame = 0;
    btnPrevImg.setEnabled(false);
    btnNextImg.setEnabled(true);

    RefreshConfigPanel();
}

// LoopHandler responds to the clicking of any of the loop menu items or
// any of the NUMPAD number accelerator keys. It changes CurrentDispLoop as
// appropriate then sets CurrentDispFrame to 0. The display will respond
// appropriately when the timer fires.

private void LoopHandler(java.awt.event.ActionEvent evt) {

    switch (evt.getActionCommand()) {

        case "1":
            CurrentDispLoop = 0;
            break;
        case "2":
            CurrentDispLoop = 1;
    }
}

```

```

        break;

    case "3":
        CurrentDispLoop = 2;
        break;

    case "4":
        CurrentDispLoop = 3;
        break;

    case "5":
        CurrentDispLoop = 4;
        break;

    case "6":
        CurrentDispLoop = 5;
        break;

    case "7":
        CurrentDispLoop = 6;
        break;

    case "8":
        CurrentDispLoop = 7;
        break;

    case "9":
        CurrentDispLoop = 8;
        break;
    }

    CurrentDispFrame = 0;
}

// SpeedHandler responds to the clicking of any of the speed menu items or
// any of the appropriate NUMPAD accelerator keys. It changes LoopInterval,
// removes the LOOPER actionlistener from the timer, kills the timer, then
// creates and fires a new timer with LOOPER and appropriate speed values.
// Note that when the . key is pressed, the timer is just stoped and if
// ENTER is pressed, the timer resumes - without a new timer being created.

private void SpeedHandler(java.awt.event.ActionEvent evt) {
    switch (evt.getActionCommand()) {

        case "FAST":
            LoopInterval -= 500;
            if (LoopInterval < 1000) {
                LoopInterval = 1000;
            }
            myTimer.removeActionListener(LOOPER);
            myTimer = null;
            myTimer = new javax.swing.Timer(LoopInterval, LOOPER);
            myTimer.start();
            break;
    }
}

```

```

case "SLOW":
    LoopInterval += 500;
    if (LoopInterval > 30000) {
        LoopInterval = 30000;
    }
    myTimer.removeActionListener(LOOPER);
    myTimer = null;
    myTimer = new javax.swing.Timer(LoopInterval, LOOPER);
    myTimer.start();
    break;

case "STOP":
    myTimer.stop();
    break;

case "RESUME":
    myTimer.restart();
    break;

case "FWD":
    myTimer.stop();
    CurrentDispFrame++;
    if (CurrentDispFrame > 24) {
        CurrentDispFrame = 0;
    }
    String FN1 = "L" + Integer.toString(CurrentDispLoop) + "F" +
                Integer.toString(CurrentDispFrame) + ".CMR";
    // Check to see if the file containing the requested image is
    // available.
    Boolean avail1 = ImageAvailable(FN1);
    // If not available, step through the loop until an image is
    // found.
    while (!avail1) {
        CurrentDispFrame++;
        if( CurrentDispFrame > 24) {
            CurrentDispFrame = 0;
        }
        FN1 = "L" + Integer.toString(CurrentDispLoop) + "F" +
              Integer.toString(CurrentDispFrame) + ".CMR";
        avail1 = ImageAvailable(FN1);
    }
    // Once an image is found to display, read the file, set
    // the image into DRetrievedImage, then display the name of
    // the camera into the status panel.

try {
    DRetrievedImage = ImageIO.read(new File(FN1));
}
catch (Throwable t) { }

```

```

        StatusLabel.setText(LatestConfig.Loops[CurrentDispLoop].
            Cameras[CurrentDispFrame].imageName);

        break;

    case "BWD":
        myTimer.stop();
        CurrentDispFrame--;
        if (CurrentDispFrame < 0) {
            CurrentDispFrame = 24;
        }
        String FN2 = "L" + Integer.toString(CurrentDispLoop) + "F" +
            Integer.toString(CurrentDispFrame) + ".CMR";
        // Check to see if the file containing the requested image is
        // available.
        Boolean avail2 = ImageAvailable(FN2);
        // If not available, step through the loop until an image is
        // found.
        while (!avail2) {
            CurrentDispFrame--;
            if( CurrentDispFrame <0 ) {
                CurrentDispFrame = 24;
            }
            FN2 = "L" + Integer.toString(CurrentDispLoop) + "F" +
                Integer.toString(CurrentDispFrame) + ".CMR";
            avail2 = ImageAvailable(FN2);
        }
        // Once an image is found to display, read the file, set
        // the image into DRetrievedImage, then display the name of
        // the camera into the status panel.
        try {
            DRetrievedImage = ImageIO.read(new File(FN2));
        }
        catch (Throwable t) { }

        StatusLabel.setText(LatestConfig.Loops[CurrentDispLoop].
            Cameras[CurrentDispFrame].imageName);

        break;
    }
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    //editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
    /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
     * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
     */
    try {
        for (javax.swing.UIManager.LookAndFeelInfo info :
            javax.swing.UIManager.getInstalledLookAndFeels()) {

```

```

        if ("Nimbus".equals(info.getName())) {
            javax.swing.UIManager.setLookAndFeel(info.getClassName());
            break;
        }
    } catch (ClassNotFoundException ex) {
        java.util.logging.Logger.getLogger(MainPanel.class.getName()).log(java.util.logging.Level.SEVERE,
                null, ex);
    } catch (InstantiationException ex) {
        java.util.logging.Logger.getLogger(MainPanel.class.getName()).log(java.util.logging.Level.SEVERE,
                null, ex);
    } catch (IllegalAccessException ex) {
        java.util.logging.Logger.getLogger(MainPanel.class.getName()).log(java.util.logging.Level.SEVERE,
                null, ex);
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
        java.util.logging.Logger.getLogger(MainPanel.class.getName()).log(java.util.logging.Level.SEVERE,
                null, ex);
    }
    //

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            new MainPanel().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JPanel CameraPanel;
private javax.swing.JPanel ConfigurePanel;
private javax.swing.JPanel FramePanel;
private javax.swing.JLabel StatusLabel;
private javax.swing.JPanel StatusPanel;
private javax.swing.ButtonGroup bgLoopConfigure;
private javax.swing.JButton btnNextImg;
private javax.swing.JButton btnPrevImg;
private javax.swing.JButton btnSave;
private javax.swing.JButton btnTest;
private javax.swing.JTextField edLoopName;
private javax.swing.JTextField edURL;
private javax.swing.JMenuBar jMenuBar2;
private javax.swing.JPopupMenu.Separator jSeparator2;
private javax.swing.JPopupMenu.Separator jSeparator3;
private javax.swing.JLabel lblFrame;
private javax.swing.JTextField lblFrameName;
private javax.swing.JLabel lblFrameNum;
private javax.swing.JLabel lblLoop;
private javax.swing.JLabel lblURL;
private javax.swing.JMenuItem mnuClose;
private javax.swing.JMenuItem mnuConfigure;
private javax.swing.JMenuItem mnuFaster;
private javax.swing.JMenu mnuFile;
private javax.swing.JMenuItem mnuLoop1;
private javax.swing.JMenuItem mnuLoop2;
private javax.swing.JMenuItem mnuLoop3;
private javax.swing.JMenuItem mnuLoop4;
private javax.swing.JMenuItem mnuLoop5;
private javax.swing.JMenuItem mnuLoop6;
private javax.swing.JMenuItem mnuLoop7;
private javax.swing.JMenuItem mnuLoop8;
private javax.swing.JMenuItem mnuLoop9;
private javax.swing.JMenu mnuLoopSelect;
private javax.swing.JMenu mnuLoopSpeed;
private javax.swing.JMenuItem mnuLooper;
private javax.swing.JMenuItem mnuReset;
private javax.swing.JMenuItem mnuResume;
private javax.swing.JMenuItem mnuSlower;
private javax.swing.JMenuItem mnuStepBwd;
private javax.swing.JMenuItem mnuStepFwd;
private javax.swing.JMenuItem mnuStop;
private javax.swing.JPanel pnlCam;
private javax.swing.JPanel pnlTestCam;
private javax.swing.JRadioButton rbLoop1;
private javax.swing.JRadioButton rbLoop2;
private javax.swing.JRadioButton rbLoop3;
private javax.swing.JRadioButton rbLoop4;
private javax.swing.JRadioButton rbLoop5;

```

```

    private javax.swing.JRadioButton rbLoop6;
    private javax.swing.JRadioButton rbLoop7;
    private javax.swing.JRadioButton rbLoop8;
    private javax.swing.JRadioButton rbLoop9;
    // End of variables declaration
}

```

FetchImagery.java:

```

package WxCam;

import java.awt.image.BufferedImage;
import java.io.File;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Calendar;
import java.util.Date;
import javax.imageio.ImageIO;

/**
 * FetchImagery is a class that is designed to run as a background thread that
 * continuously cycles through an instance of WxCamConfig and fetches images for
 * storage on disk that have not been fetched for over 5 minutes. The process
 * runs through the cycle, then sleeps for one minute before repeating the
 * process. The sleep/resume process increases system performance and decreases
 * disk activity.
 *
 * @author Jay Albrecht, National Weather Service Seattle, July 2012.
 */

public class FetchImagery extends Thread {
    private Boolean stopDownload = false;           // run until endThread() is called

    @Override
    public void run() {
        while (!stopDownload) {
            // Get a copy of WxCamConfig from disk.
            WxCamConfig Z = WxCamConfig.getConfigFile();

            // For each loop in the configuration file:
            for (int LOOP = 0; LOOP < 9; LOOP++) {
                if (Z.Loops[LOOP].LoopName.trim().equals("")) {
                    continue;          // Ignore the whole loop if there
                                         // is no loop name.
                }

                // For each frame in the loop:
                for (int FRAME = 0; FRAME < 25; FRAME++) {
                    if (Z.Loops[LOOP].Cameras[FRAME].imageName.trim().
                        equals("")) {
                        continue;      // Ignore the camera if there is no
                                         // camera name.
                    }

                    String myURL = Z.Loops[LOOP].Cameras[FRAME].imageURL.trim();
                    // Construct imagename based on loop and frame. The name of
                    // a downloaded image takes the form L#F##.CMR where the
                    // first number is the loop number - 1 and the second number
                    // is the camera number - 1 (zero based convention of Java).

                    String FN = "L" + Integer.toString(LOOP) + "F" +
                               Integer.toString(FRAME) + ".CMR";
                }
            }
        }
    }
}

```

```

// avail is a boolean that states whether or not the image
// with the name FN is stored on disk. old is a boolean
// that states whether the image is more than 5 minutes old.
// isGoodURL is a boolean that states whether the URL for
// the camera object defined by the CameraDef in the
// configuration file is a valid URL.

// If an image is not on disk or is old -- and the URL is
// valid, the image is downloaded and saved to disk in jpg
// format.

Boolean avail = ImageAvailable(FN);

Boolean old = false;

if (avail == true) {

    old = ImageOlderThan5Min(FN);

}

Boolean isGoodURL = goodURL(myURL);

if (((avail == false) ||
     (old == true)) && isGoodURL == true) {

    // Retrieve the image from the URL of the CameraDef object,
    // then write the image as a jpg file to the file FN.

    try {

        BufferedImage retrieved = ImageIO.read(
            new URL(myURL));
        ImageIO.write(retrieved, "jpg", new File(FN));

    }

    catch (Throwable t) {}

}

}

// Now that the thread has processed all of the cameras in all of
// the loops, sleep for 60 seconds.

try {

    FetchImagery.sleep(60000);

}

catch (Throwable t) {}

}

// endThread stops the downloading of imagery by setting the class variable
// stopDownload to true...which ends the thread's while loop.

public void endThread() {

    stopDownload = true;
}

// ImageAvailable(String ImageName) determines whether the filename
// ImageName is stored on the disk in the application directory. If it
// exists, the method returns true, otherwise it returns false.

private Boolean ImageAvailable(String ImageName) {

    Boolean isAvailable;

```

```

        File file = new File(FileName);
        isAvailable = file.exists();

        return isAvailable;
    }

    // ImageOlderThan5Min(String ImageName) determines the age of the file
    // ImageName and determines if it is older than 5 minutes. If so, it returns
    // true, otherwise it returns false.

    private Boolean ImageOlderThan5Min(String ImageName) {

        // Retrieve the modification date/time of the file ImageName

        File file = new File(FileName);
        Calendar FDate = Calendar.getInstance();
        FDate.setTime(new Date(file.lastModified()));

        // Get the current date and time, then add 5 minutes.

        Calendar NDate = Calendar.getInstance();
        FDate.add(Calendar.MINUTE, 5);

        if (FDate.after(NDate)) {

            return false;
        }

        else {

            return true;
        }
    }

    // goodURL(String stURL) tests the string stURL to determine if it can be
    // resolved as a URL object. If it can, it returns true, otherwise it
    // returns false.

    private Boolean goodURL(String stURL) {

        try {

            URL url = new URL(stURL);
            return true;
        }

        catch (MalformedURLException e) {

            return false;
        }
    }
}

```

WxCamConfig.java:

```

package WxCam;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

/**

```

```

* WxCamConfig is a class that is used to define a full program configuration
* that holds 9 loops that can contain 25 images each. A WxCamConfig object only
* contains 9 LoopDef objects.
*
* The LoopDef array member variable is public. The class has two constructors.
* The first sets the member LoopDef array members as null. The second
* constructor is a copy constructor. The class also defines a method that
* retrieves a saved WxCamConfig file and one that saves a WxCamConfig file.
*
* WxCamConfig implements Serializable so that WxCam configurations can bundle
* LoopDef objects and CameraDef objects without issue.
*
* @author Jay Albrecht, National Weather Service, Seattle WA July 2012.
*/

```

```

public class WxCamConfig implements Serializable{
    public LoopDef[] Loops = new LoopDef[9];      // Array of up to 9 LoopDef
                                                   // objects

    // Main WxCamConfig constructor. Creates 9 null LoopDef objects, each which
    // will contain 25 null CameraDef objects.

    public WxCamConfig() {
        for (int i = 0; i < 9; i++) {
            this.Loops[i] = new LoopDef();
        }
    }

    // Copy constructor.
    public WxCamConfig(WxCamConfig SC) {
        for (int i = 0; i < 9; i++) {
            this.Loops[i] = new LoopDef();
            this.Loops[i] = SC.Loops[i];
        }
    }

    // method used to retrieve a WxCamConfig file from disk. The file is stored
    // in the same location as the application and has the file extension .MCF.
    // The configuration file is returned as a WxCamConfig object. If one cannot
    // be found on disk, an empty WxCamConfig object is returned.

    public static WxCamConfig getConfigFile() {
        WxCamConfig Z;
        // try to read the configuration file in from disk...
        // if not found or successfully read, return an empty WxCamConfig object.

        ObjectInputStream in;
        try {
            in = new ObjectInputStream(new FileInputStream(
                "WxCamConfig.MCF"));
            Z = (WxCamConfig) in.readObject();
            in.close();
        }
        return Z;
    }

    catch (ClassNotFoundException | IOException e) {
        return new WxCamConfig(); }
    }

    // Try to save a configuration file from WxCamConfig object Z. If the

```

```

// save process fails, nothing more than a System.out command is executed,
// and that will likely not be seen by the user.

public static void saveConfigFile(WxCamConfig Z) {
    try{
        FileOutputStream saveFile=new FileOutputStream("WxCamConfig.MCF");
        try (ObjectOutputStream save = new ObjectOutputStream(saveFile)) {
            save.writeObject(Z);
        }
    }
    catch(Exception exc){
        System.out.println("Save failed");
    }
}
}

```

LoopDef.java:

```

package WxCam;

import java.io.Serializable;

/**
 * LoopDef is a class that is used to define a set of up to 25 images that will
 * be looped by the WxCam application. A LoopDef consists of a String LoopName
 * value that simply gives the loop a name and an array of 25 CameraDef objects.
 *
 * LoopDef member variables are public. The class is quite simple containing
 * only constructors (an empty constructor that assigns null values to the loop
 * name and CameraDef array, a constructor that assigns a loop name but null
 * values to the CameraDef array, and a copy constructor).
 *
 * LoopDef implements Serializable so that WxCam configurations can bundle
 * LoopDef objects and CameraDef objects without issue.
 *
 * @author Jay Albrecht, National Weather Service, Seattle WA July 2012.
 */

public class LoopDef implements Serializable{

    public String LoopName = ""; // Name of the Loop
    public CameraDef[] Cameras = new CameraDef[25]; // Array of up to 25
                                                    // CameraDef objects

    // Empty constructor. Sets null values for the name of the loop and for
    // 25 CameraDef objects that will comprise the loop.

    public LoopDef() {
        this.LoopName = "";
        for (int i = 0; i < 25; i++) {
            this.Cameras[i] = new CameraDef();
        }
    }

    // Constructor that sets a passed String Loop name. This constructor sets
    // null values for the 25 CameraDef objects that will comprise the loop.

    public LoopDef(String N) {

```

```

        this.LoopName = N;
        for (int i = 0; i < 25; i++) {
            this.Cameras[i] = new CameraDef();
        }
    }
    // Copy constructor.
    public LoopDef(LoopDef LD){
        this.LoopName = LD.LoopName;
        System.arraycopy(LD.Cameras, 0, this.Cameras, 0, 25);
    }
}

```

CameraDef.java:

```

package WxCam;
import java.io.Serializable;

/**
 * CameraDef is simply the definition of a weather camera to be followed; it
 * includes the URL of the image location and a String name of the image that is
 * used to define the image on a label placed on a status panel.
 *
 * CameraDef member variables are public. The class is quite simple containing
 * only constructors (an empty constructor that assigns null values to the the
 * image location and name, a constructor that contains a URL and image name,
 * and a copy constructor).
 *
 * CameraDef implements Serializable so that loop definitions and WxCam
 * configurations can bundle CameraDef objects without issue.
 *
 * @author Jay Albrecht, National Weather Service, Seattle WA July 2012.
 */
public class CameraDef implements Serializable {

    public String imageURL = "";           // URL location of image as a String
    public String imageName = "";          // Name of an image as a String

    // Empty Constructor used to set null values to the image URL and name.
    public CameraDef() {
        this.imageURL = "";
        this.imageName = "";
    }

    // Constructor used to create a CameraDef given a URL and an image name.
    public CameraDef(String U, String N) {
        this.imageURL = U;
        this.imageName = N;
    }

    // Copy constructor
    public CameraDef(CameraDef C) {
        this.imageURL = C.imageURL;
        this.imageName = C.imageName;
    }
}

```